Scalable Verification of Markov Decision Processes

Axel Legay, Sean Sedwards and Louis-Marie Traonouez

Inria Rennes – Bretagne Atlantique

Abstract Markov decision processes (MDP) are useful to model concurrent process optimisation problems, but verifying them with numerical methods is often intractable. Existing approximative approaches do not scale well and are limited to memoryless schedulers. Here we present the basis of scalable verification for MDPSs, using an $\mathcal{O}(1)$ memory representation of history-dependent schedulers. We thus facilitate scalable learning techniques and the use of massively parallel verification.

1 Introduction

Markov decision processes (MDP) describe systems that interleave nondeterministic *actions* and probabilistic transitions, possibly with rewards or costs assigned to the actions [3,19]. This model has proved useful in many real optimisation problems and may also be used to represent concurrent probabilistic programs (see, e.g., [4,2]). Such models comprise probabilistic subsystems whose transitions depend on the states of the other subsystems, while the order in which concurrently enabled transitions execute is nondeterministic. This order may radically affect the expected reward or the probability that a system will satisfy a given property. It is therefore useful to calculate the upper and lower bounds of these quantities.

Fig. 1 shows a typical fragment of an MDP. Referring in parentheses to the labels in the figure, the execution semantics are as follows. In a given state (s_0), an action ($a_1, a_2, \ldots$) is chosen nondeterministically to select a distribution of probabilistic transitions ($p_1, p_2, \ldots$ or p_3, p_4, etc.). A probabilistic choice is then made to select the next state ($s_1, s_2, s_3, s_4, \ldots$). To each of the actions may be associated a reward ($r_1, r_2, \ldots$), allowing values to be assigned to sequences of actions.

To calculate the expected total reward or the expected probability of a sequence of states, it is necessary to define how the nondeterminism in the MDP will be resolved. In the literature this is often called a *strategy*, a *policy* or an *adversary*. Here we use the term *scheduler* and focus on MDPs in the context of *model checking* concurrent probabilistic systems. Model checking is an automatic technique to verify that a system satisfies a property specified in temporal logic [7]. *Probabilistic* model checking quantifies the probability that a probabilistic system will satisfy a property [9]. Classic analysis of MDPs is concerned with finding schedulers that maximise or minimise rewards [3,19]. The classic verification algorithms for MDPs are concerned with finding schedulers that maximise

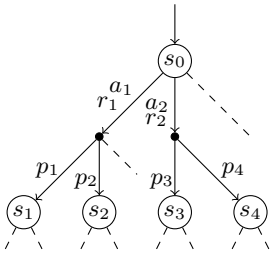

Figure 1. Fragment of a typical Markov decision process.

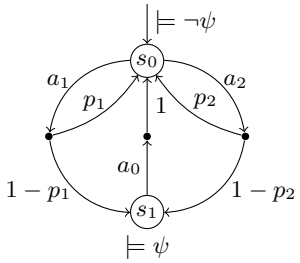

Figure 2. MDP with different optima for general and memoryless schedulers when $p_1 \neq p_2$.

or minimise the probability of a property, or deciding the existence of a scheduler that ensures the probability of a property is within some bound [4]. Our techniques can be easily extended to include rewards, but in this work we focus on probabilities and leave rewards for future consideration.

1.1 Schedulers and State Explosion

The classic algorithms to solve MDPs are *policy iteration* and *value iteration* [19]. Model checking algorithms for MDPs may use value iteration applied to probabilities [2, Ch. 10] or solve the same problem using linear programming [4]. All consider *history-dependent* schedulers. Given an MDP with set of actions A, having a set of states S that induces a set of sequences of states $\Omega = S^+$, a history-dependent (general) scheduler is a function $\mathfrak{S} : \Omega \to A$. A memoryless scheduler is a function $\mathfrak{M} : S \to A$. Intuitively, at each state in the course of an execution, a history-dependent scheduler ($\mathfrak{S}$) chooses an action based on the sequence of previous states, while a memoryless scheduler ($\mathfrak{M}$) chooses an action based only on the current state. History-dependent schedulers therefore include memoryless schedulers.

Fig. 2 illustrates a simple MDP for which memoryless and history-dependent schedulers give different optima for logical property $\mathbf{X}(\psi \wedge \mathbf{X}\mathbf{G}^t \neg\psi)$ when $p_1 \neq p_2$ and $t > 0$. The property makes use of the temporal operators *next* ($\mathbf{X}$) and *globally* ($\mathbf{G}$). Intuitively, the property states that on the next step ψ will be true and, on the step after that, $\neg\psi$ will be remain true for $t+1$ time steps. The property is satisfied by the sequence of states $s_0 s_1 s_0 s_0 \cdots$. If $p_1 > p_2$, the maximum probability for $s_0 s_1$ is achieved with action a_2, while the maximum probability for $s_0 s_0$ is achieved with action a_1. Given that both transitions start in the same state, a memoryless scheduler will not achieve the maximum probability achievable with a history-dependent scheduler.

The principal challenge of finding optimal schedulers is what has been described as the 'curse of dimensionality' [3] and the 'state explosion problem' [7]: the number of states of a system increases exponentially with respect to the number of interacting components and state variables. This phenomenon has

led to the design of sampling algorithms that find 'near optimal' schedulers to maximise rewards in discounted MDPs. Probably the best known is the Kearns algorithm [13], which we briefly review in Section 2.

The state explosion problem of model checking applied to purely probabilistic systems has been well addressed by *statistical* model checking (SMC) [21]. SMC uses an executable model to approximate the probability that a system satisfies a specified property by the proportion of simulation traces that individually satisfy it. SMC algorithms work by constructing an automaton to accept only traces that satisfy the property. This automaton may then be used to estimate the probability of the property or to decide an hypothesis about the probability. Typically, the probability of property φ is estimated by $\frac{1}{N}\sum_{i=1}^{N}\mathbf{1}(\omega_i \models \varphi)$, where $\omega_1, \ldots, \omega_N$ are N independently generated simulation traces and $\mathbf{1}(\cdot)$ is an indicator function that corresponds to the output of the automaton: it returns 1 if the trace is accepted and 0 if it is not. N is chosen a priori to give the required statistical confidence (e.g., using a Chernoff bound [18], see Section 4.2). Sequential hypothesis tests (e.g., Wald's sequential probability ratio test [20], see Section 4.1) do not define N a priori, but generate simulation traces until an hypothesis can be accepted or rejected with specified confidence. The state space of the system is not constructed explicitly–states are generated on the fly during simulation–hence SMC is efficient for large, possibly infinite state, systems. Moreover, since the simulations are required to be statistically independent, SMC may be easily and efficiently divided on parallel computing architectures.

SMC cannot be applied to MDPs without first resolving the nondeterminism. Since nondeterministic and probabilistic choices are interleaved in an MDP, schedulers are typically of the same order of complexity as the system as a whole and may be infinite. As a result, existing SMC algorithms for MDPs consider only memoryless schedulers and have other limitations (see Section 2).

1.2 Our Approach

We have created memory-efficient techniques to facilitate Monte Carlo verification of nondeterministic systems, *without* storing schedulers explicitly. In essence, the possibly infinite behaviour of schedulers is fully specified *implicitly* by the seed of a pseudo-random number generator. Our techniques therefore require almost no additional memory over standard SMC. In doing this, we are the first to provide the basis for a complete lightweight statistical alternative to the standard numerical verification algorithms for MDPs. A further contribution is our derivation of the statistical confidence bounds necessary to test multiple schedulers. These results suggest obvious solutions to problems encountered with existing algorithms that rely on multiple statistical tests (e.g., [11]).

In this work we demonstrate the core ideas of our approach with simple SMC algorithms that repeatedly sample from scheduler space. Practical implementations require more sophisticated algorithms that adopt "smart sampling" (optimal use of simulation budget) and lightweight learning techniques. Some of our results make use of these ideas, but a full exposition is not possible here.

2 Related Work

The Kearns algorithm [13] is the classic 'sparse sampling algorithm' for large, infinite horizon, discounted MDPs. It constructs a 'near optimal' scheduler piecewise, by approximating the best action from a current state using a stochastic depth-first search. Importantly, optimality is with respect to rewards, not probability (as required by standard model checking tasks). The algorithm can work with large, potentially infinite state MDPs because it explores a probabilistically bounded search space. This, however, is exponential in the discount. To find the action with the greatest expected reward in the current state, the algorithm recursively estimates the rewards of successive states, up to some maximum depth defined by the discount and desired error. Actions are enumerated while probabilistic choices are explored by sampling, with the number of samples set as a parameter. The error is specified as a maximum difference between consecutive estimates, allowing the discount to guarantee that the algorithm will eventually terminate.

There have been several recent attempts to apply SMC to nondeterministc models [5,16,11,10]. In [5,10] the authors present on-the-fly algorithms to remove 'spurious' nondeterminism, so that standard SMC may be used. This approach is limited to the class of models whose nondeterminism does not affect the resulting probability of a property–scheduling makes no difference. The algorithms therefore do not attempt to address the standard MDP model checking problems related to finding optimal schedulers.

In [16] the authors first find a memoryless scheduler that is near optimal with respect to a reward scheme and discount, using an adaptation of the Kearns algorithm. This induces a Markov chain whose properties may be verified with standard SMC. By storing and re-using information about visited states, the algorithm improves on the performance of the Kearns algorithm, but is thus limited to memoryless schedulers that fit into memory. The near optimality of the induced Markov chain is with respect to rewards, not probability, hence [16] does not address the standard model checking problems of MDPs.

In [11] the authors present an SMC algorithm to decide whether there exists a memoryless scheduler for a given MDP, such that the probability of a property is above a given threshold. The algorithm has an inner loop that generates candidate schedulers by iteratively improving a probabilistic scheduler according to sample traces that satisfy the property. The algorithm is limited to memoryless schedulers because the improvement process counts state-action pairs. The outer loop tests the candidate scheduler against the hypothesis using SMC and is iterated until an example is found or sufficient attempts have been made. The inner loop does not in general converge to the true optimum, but the outer loop randomly explores local maxima. This makes the number of samples used by the inner loop critical: too many may significantly reduce the scope of the random exploration and thus reduce the probability of finding the global optimum. A further problem is that the repeated hypothesis tests of the outer loop will eventually produce erroneous results. We address this phenomenon in Section 4.

We conclude that (*i*) no previous approach is able to provide a complete set of SMC algorithms for MDPs, (*ii*) no previous SMC approach considers history-dependent schedulers and (*iii*) no previous approach facilitates light-weight sampling from scheduler space.

3 Schedulers as Seeds of Random Number Generators

Storing schedulers as explicit mappings does not scale, so we have devised a way to represent schedulers using uniform pseudo-random number generators (PRNG) that are initialised by a *seed* and iterated to generate the next pseudo-random value. In general, such PRNGs aim to ensure that arbitrary subsets of sequences of iterates are uniformly distributed and that consecutive iterates are statistically independent. PRNGs are commonly used to implement the uniform probabilistic scheduler, which chooses actions uniformly at random and thus explores all possible combinations of nondeterministic choices. Executing such an implementation twice with the same seed will produce identical traces. Executing the implementation with a different seed will produce an unrelated set of choices. Individual deterministic schedulers cannot be identified, so it is not possible to estimate the probability of a property under a specific scheduler.

An apparently plausible solution is to use independent PRNGs to resolve nondeterministic and probabilistic choices. It is then possible to generate multiple probabilistic simulation traces per scheduler by keeping the seed of the PRNG for nondetermistic choices fixed while choosing random seeds for a separate PRNG for probabilistic choices. Unfortunately, the schedulers generated by this approach do not span the full range of general or even memoryless schedulers. Since the sequence of iterates from the PRNG used for nondeterministic choices will be the same for all instantiations of the PRNG used for probabilistic choices, the i^{th} iterate of the PRNG for nondeterministic choices will always be the same, regardless of the state arrived at by the previous probabilistic choices. The i^{th} chosen action can be neither state nor trace dependent.

3.1 General Schedulers Using Hash Functions

Our solution is to construct a per-step PRNG seed that is a *hash* of the an integer identifying a specific scheduler concatenated with an integer representing the sequence of states up to the present.

We assume that a state of an MDP is an assignment of values to a vector of system variables $v_i, i \in \{1, \ldots, n\}$. Each v_i is represented by a number of bits b_i, typically corresponding to a primitive data type (*int*, *float*, *double*, etc.). The state can thus be represented by the concatenation of the bits of the system variables, such that a sequence of states may be represented by the concatenation of the bits of all the states. Without loss of generality, we interpret such a sequence of states as an integer of $\sum_{i=1}^{n} b_i$ bits, denoted $\overline{s}$, and refer to this in general as the *trace vector*. A scheduler is denoted by an integer σ, which is concatenated to $\overline{s}$ (denoted $\sigma : \overline{s}$) to uniquely identify a trace and a scheduler.

Our approach is to generate a hash code $h = \mathcal{H}(\sigma : \overline{s})$ and to use h as the seed of a PRNG that resolves the next nondeterministic choice.

The hash function $\mathcal{H}$ thus maps $\sigma : \overline{s}$ to a seed that is deterministically dependent on the trace and the scheduler. The PRNG maps the seed to a value that is uniformly distributed but nevertheless deterministically dependent on the trace and the scheduler. In this way we approximate the scheduler functions $\mathfrak{S}$ and $\mathfrak{M}$ described in Section 1.1. Importantly, our technique only relies on the standard properties of hash functions and PRNGs. Algorithm 1 is the basic simulation function of our algorithms.

Algorithm 1: Simulate

Input:
$\mathcal{M}$: an MDP with initial state s_0
φ: a property
σ: an integer identifying a scheduler
Output:
ω: a simulation trace

Let $\mathcal{U}_{\text{prob}}, \mathcal{U}_{\text{nondet}}$ be uniform PRNGs with respective samples $r_{\text{pr}}, r_{\text{nd}}$
Let $\mathcal{H}$ be a hash function
Let s denote a state, initialised $s \leftarrow s_0$
Let ω denote a trace, initialised $\omega \leftarrow s$
Let $\overline{s}$ be the trace vector, initially empty
Set seed of $\mathcal{U}_{\text{prob}}$ randomly
while $\omega \models \varphi$ *is not decided* **do**
  $\overline{s} \leftarrow \overline{s} : s$
  Set seed of $\mathcal{U}_{\text{nondet}}$ to $\mathcal{H}(\sigma : \overline{s})$
  Iterate $\mathcal{U}_{\text{nondet}}$ to generate r_{nd} and use to resolve nondeterministic choice
  Iterate $\mathcal{U}_{\text{prob}}$ to generate r_{pr} and use to resolve probabilistic choice
  Set s to the next state
  $\omega \leftarrow \omega : s$

3.2 An Efficient Iterative Hash Function

To implement our approach, we have devised an efficient hash function that constructs seeds incrementally. The function is based on modular division [14, Ch. 6], such that $h = (\sigma : \overline{s}) \bmod m$, where m is a suitably large prime.

Since $\overline{s}$ is a concatenation of states, it is usually very much larger than the maximum size of integers supported as primitive data types. Hence, to generate h we use Horner's method [12][14, Ch. 4]: we set $h_0 = \sigma$ and find $h \equiv h_n$ (n as given in Section 3.1) by iterating the recurrence relation

$$h_i = (h_{i-1} 2^{b_i} + v_i) \bmod m. \tag{1}$$

The size of m defines the maximum number of different hash codes. The precise value of m controls how the hash codes are distributed. To avoid collisions,

a simple heuristic is that m should be a large prime not close to a power of 2 [8, Ch. 11]. Practically, it is an advantage to perform calculations using primitive data types that are native to the computational platform, so the sum in (1) should be less than or equal to the maximum permissible value. To achieve this, given $x, y, m \in \mathbb{N}$, we note the following congruences:

$$(x + y) \bmod m \equiv (x \bmod m + y \bmod m) \bmod m \tag{2}$$

$$(xy) \bmod m \equiv ((x \bmod m)(y \bmod m)) \bmod m \tag{3}$$

The addition in (1) can thus be re-written in the form of (2), such that each term has a maximum value of $m - 1$:

$$h_i = ((h_{i-1}2^{b_i}) \bmod m + (v_i) \bmod m) \bmod m \tag{4}$$

To prevent overflow, m must be no greater than half the maximum possible integer. Re-writing the first term of (4) in the form of (3), we see that before taking the modulus it will have a maximum value of $(m-1)^2$, which will exceed the maximum possible integer. To avoid this, we take advantage of the fact that h_{i-1} is multiplied by a power of 2 and that m has been chosen to prevent overflow with addition. We thus apply the following recurrence relation:

$$(h_{i-1}2^j) \bmod m = (h_{i-1}2^{j-1}) \bmod m + (h_{i-1}2^{j-1}) \bmod m \tag{5}$$

Equation (5) allows our hash function to be implemented using efficient native arithmetic. Moreover, we infer from (1) that to find the hash code corresponding to the current state in a trace, we need only know the current state and the hash code from the previous step. When considering memoryless schedulers we need only know the current state.

4 Confidence with Multiple Estimates

The Chernoff bound [18,6] and Wald sequential probability ratio test [20] are commonly used to bound errors of SMC algorithms. Their guarantees are probabilistic, such that with specified non-zero probability they produce an incorrect result. If such bounds are used on M schedulers, some of whose true probabilities lie in the interval $(0, 1)$, then as $M \to \infty$ the probability of encountering an error is a.s. 1. In particular, the maximum and minimum estimates will tend to 1 and 0, respectively, regardless of the true values.

To overcome this phenomenon, in Sects. 4.1 and 4.2 we derive new confidence bounds to allow SMC algorithms to test multiple schedulers. We illustrate their use with simple algorithms that sample M schedulers at random, where M is a parameter. These algorithms are the basis of a technique we call "smart sampling", which can exponentially improve convergence. The basic idea is to assign part of the simulation budget to obtain a coarse estimate of the extremal probabilities and to use this information to generate a set of schedulers that contains a "good" scheduler with high probability. The remaining budget is

used to refine the set to find the best scheduler. Smart sampling has provided improvements of several orders of magnitude with the illustrated examples and is the subject of ongoing development. Lack of space prevents further discussion.

4.1 Sequential Probability Ratio Test for Multiple Schedulers

The sequential probability ratio test (SPRT) of Wald [20] evaluates hypotheses of the form $\mathrm{P}(\omega \models \varphi) \bowtie p$, where $\bowtie \in \{\leq, \geq\}$. The SPRT distinguishes between two hypotheses, $H_0 : \mathrm{P}(\omega \models \varphi) \geq p^0$ and $H_1 : \mathrm{P}(\omega \models \varphi) \leq p^1$, where $p^0 > p^1$. Hence, to evaluate $\mathrm{P}(\omega \models \varphi) \bowtie p$, the SPRT requires a region of indecision (an 'indifference region' [21]) which may be specified by parameter ϑ, such that $p^0 = p + \vartheta$ and $p^1 = p - \vartheta$. The SPRT also requires parameters α and β, which specify the maximum acceptable probabilities of errors of the first and second kind, respectively. An error of the first kind is incorrectly rejecting a true H_0; an error of the second kind is incorrectly accepting a false H_0. To choose between H_0 and H_1, the SPRT defines the probability ratio

$$\mathit{ratio} = \prod_{i=1}^{n} \frac{(p^1)^{\mathbf{1}(\omega_i \models \varphi)}(1-p^1)^{\mathbf{1}(\omega_i \not\models \varphi)}}{(p^0)^{\mathbf{1}(\omega_i \models \varphi)}(1-p^0)^{\mathbf{1}(\omega_i \not\models \varphi)}},$$

where n is the number of simulation traces ω_i, $i \in \{1, \ldots, n\}$, generated so far. The test proceeds by performing a simulation and calculating *ratio* until one of two conditions is satisfied: H_1 is accepted if $\mathit{ratio} \geq (1-\beta)/\alpha$ and H_0 is accepted if $\mathit{ratio} \leq \beta/(1-\alpha)$.

To decide whether there exists a scheduler such that $\mathrm{P}(\omega \models \varphi) \bowtie p$, we would like to apply the SPRT to multiple (randomly chosen) schedulers. The idea is to test different schedulers, up to some specified number M, until an example is found. Since the probability of error with the SPRT applied to multiple hypotheses is cumulative, we consider the probability of no errors in any of M tests. Hence, in order to ensure overall error probabilities α and β, we adopt $\alpha_M = 1 - \sqrt[M]{1-\alpha}$ and $\beta_M = 1 - \sqrt[M]{1-\beta}$ in our stopping conditions. H_1 is accepted if $\mathit{ratio} \geq (1-\beta_M)/\alpha_M$ and H_0 is accepted if $\mathit{ratio} \leq \beta_M/(1-\alpha_M)$. Algorithm 2 demonstrates the sequential hypothesis test for multiple schedulers. If the algorithm finds an example, the hypothesis is true with at least the specified confidence.

4.2 Chernoff Bound for Multiple Schedulers

Given that a system has true probability p of satisfying a property, the Chernoff bound ensures $\mathrm{P}(\mid \hat{p} - p \mid \geq \varepsilon) \leq \delta$, i.e., that the estimate $\hat{p}$ will be outside the interval $[p-\varepsilon, p+\varepsilon]$ with probability less than or equal to δ. Parameter δ is related to the number of simulations N by $\delta = 2e^{-2N\varepsilon^2}$ [18], giving

$$N = \left\lceil (\ln 2 - \ln \delta)/(2\varepsilon^2) \right\rceil. \tag{6}$$

Algorithm 2: Hypothesis testing with multiple schedulers

Input:
$\mathcal{M}, \varphi$: the MDP and property of interest
$H \in \{H_0, H_1\}$: the hypothesis of interest with threshold $p \pm \vartheta$
α, β: the desired error probabilities of H
M: the maximum number of schedulers to test
Output: The result of the hypothesis test

Let $p^0 = p + \vartheta$ and $p^1 = p - \vartheta$ be the bounds of H
Let $\alpha_M = 1 - \sqrt[M]{1-\alpha}$ and $\beta_M = 1 - \sqrt[M]{1-\beta}$
Let $A = (1-\beta_M)/\alpha_M$ and $B = \beta_M/(1-\alpha_M)$
Let $\mathcal{U}_{\text{seed}}$ be a uniform PRNG and σ be its sample
for $i \in \{1, \ldots, M\}$ *while* H *is not accepted* **do**
  Iterate $\mathcal{U}_{\text{seed}}$ to generate σ_i
  Let *ratio* $= 1$
  while *ratio* $< A \wedge$ *ratio* $> B$ **do**
    $\omega \leftarrow \text{Simulate}(\mathcal{M}, \varphi, \sigma_i)$
    ratio $\leftarrow \frac{(p^1)^{\mathbf{1}(\omega \models \varphi)}(1-p^1)^{\mathbf{1}(\omega \not\models \varphi)}}{(p^0)^{\mathbf{1}(\omega \models \varphi)}(1-p^0)^{\mathbf{1}(\omega \not\models \varphi)}}$ *ratio*
  if *ratio* $\geq A \wedge H = H_0 \vee$ *ratio* $\leq B \wedge H = H_1$ **then**
    accept H

The user specifies ε and δ and the SMC algorithm calculates N to guarantee the estimate accordingly. Equation (6) is derived from equations

$$\mathrm{P}(\hat{p} - p \geq \varepsilon) \leq e^{-2N\varepsilon^2} \qquad \text{and} \qquad \mathrm{P}(p - \hat{p} \geq \varepsilon) \leq e^{-2N\varepsilon^2}, \tag{7}$$

giving $N = \lceil (\ln \delta)/(2\varepsilon^2) \rceil$ to satisfy either inequality.

We consider the strategy of sampling M schedulers to estimate the optimum probability. We thus generate M estimates $\{\hat{p}_1, \ldots, \hat{p}_M\}$ and take either the maximum ($\hat{p}_{\max}$) or minimum ($\hat{p}_{\min}$), as required. To overcome the cumulative probability of error with the standard Chernoff bound, we specify that *all* estimates $\hat{p}_i$ must be within ε of their respective true values p_i, ensuring that any $\hat{p}_{\min}, \hat{p}_{\max} \in \{\hat{p}_1, \ldots, \hat{p}_M\}$ are within ε of their true value. Given (7) and the fact that all estimates $\hat{p}_i$ are statistically independent, the probability that all estimates are less than their upper bound is expressed by $\mathrm{P}(\bigwedge_{i=1}^{M} \hat{p}i - p_i \leq \varepsilon) \geq (1 - e^{-2N\varepsilon^2})^M$. Hence, $\mathrm{P}(\bigvee_{i=1}^{M} \hat{p}_i - p_i \geq \varepsilon) \leq 1 - (1 - e^{-2N\varepsilon^2})^M$. This leads to the following expression for N, given parameters M, ε and δ:

$$N = \left\lceil -\ln\left(1 - \sqrt[M]{1-\delta}\right)/2\varepsilon^2 \right\rceil \tag{8}$$

Since the case for $p_{\min}$ is symmetrical, (8) also ensures $\mathrm{P}(p_{\min} - \hat{p}_{\min} \geq \varepsilon) \leq \delta$. Hence, to ensure the more usual conditions that $\mathrm{P}(\mid p_{\max} - \hat{p}_{\max} \mid \geq \varepsilon) \leq \delta$ and $\mathrm{P}(\mid p_{\min} - \hat{p}_{\min} \mid \geq \varepsilon) \leq \delta$,

$$N = \left\lceil \left(\ln 2 - \ln\left(1 - \sqrt[M]{1-\delta}\right)\right)/(2\varepsilon^2) \right\rceil. \tag{9}$$

N scales logarithmically with M (e.g., for $\varepsilon = \delta = 0.01$, $N \approx \log_{1.0002}(M) + 26472$), making it tractable to consider many schedulers. Algorithm 3 is the resulting extremal probability estimation algorithm for multiple schedulers.

Algorithm 3: Extremal probability estimation with multiple schedulers

Input:
$\mathcal{M}, \varphi$: the MDP and property of interest
ε, δ: the required confidence bound
M: the number of schedulers to test
Output: Extremal estimates $\hat{p}_{\min}$ and $\hat{p}_{\max}$

Let $N = \left\lceil \ln(2/(1 - \sqrt[M]{1-\delta}))/(2\varepsilon^2) \right\rceil$ be the no. of simulations per scheduler
Let $\mathcal{U}_{\text{seed}}$ be a uniform PRNG and σ its sample
Initialise $\hat{p}_{\min} \leftarrow 1$ and $\hat{p}_{\max} \leftarrow 0$
Set seed of $\mathcal{U}_{\text{seed}}$ randomly
for $i \in \{1, \ldots, M\}$ **do**
  Iterate $\mathcal{U}_{\text{seed}}$ to generate σ_i
  Let $\mathit{truecount} = 0$ be the initial number of traces that satisfy φ
  for $j \in \{1, \ldots, N\}$ **do**
    $\omega_j \leftarrow \text{Simulate}(\mathcal{M}, \varphi, \sigma_i)$
    $\mathit{truecount} \leftarrow \mathit{truecount} + \mathbf{1}(\omega_j \models \varphi)$
  Let $\hat{p}_i = \mathit{truecount}/N$
  if $\hat{p}_{\max} < \hat{p}_i$ **then**
    $\hat{p}_{\max} = \hat{p}_i$
  if $\hat{p}_i > 0 \wedge \hat{p}_{\min} > \hat{p}_i$ **then**
    $\hat{p}_{\min} = \hat{p}_i$
if $\hat{p}_{\max} = 0$ **then**
  No schedulers were found to satisfy φ

4.3 Experiments

We implemented Algorithms 2 and 3 in our statistical model checking platform Plasma [1] and performed a number of experiments.

Figure 3 shows the empirical cumulative distribution of schedulers generated by Algorithm 3 applied to the MDP of Fig. 2, using $p_1 = 0.9$, $p_2 = 0.5$, $\varphi = \mathbf{X}(\psi \wedge \mathbf{X}\mathbf{G}^4 \neg\psi)$, $\varepsilon = 0.01$, $\delta = 0.01$ and $M = 300$. The vertical red and blue lines mark the true probabilities of φ under each of the history-dependent and memoryless schedulers, respectively. The grey rectangles show the $\pm\varepsilon$ error bounds, relative to the true probabilities. There are multiple estimates per scheduler, but all estimates are within their respective confidence bounds. Note that the confidence is specified with respect to estimates, not with respect to optimality. Defining confidence with respect to optimality remains an open problem.

In Fig. 4 we consider a reachability property of the Wireless LAN (WLAN) protocol model of [15]. The protocol aims to minimise "collisions" between devices sharing a communication channel. We estimated the probability of the

second collision at time steps $\{0, 10, \ldots, 100\}$, using Algorithm 3 with $M = 4000$ schedulers per point. Maximum and minimum estimated probabilities are denoted by blue and red circles, respectively. Maximum probabilities calculated by numerical model checking are denoted by black crosses. The shaded areas indicate the $\pm\varepsilon$ error of the estimates (Chernoff bound $\varepsilon = \delta = 0.01$) and reveal that our estimates are very close to the true values.

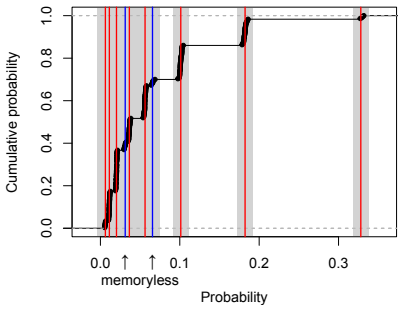

Figure 3. Empirical cumulative distribution of estimates from Algorithm 3.

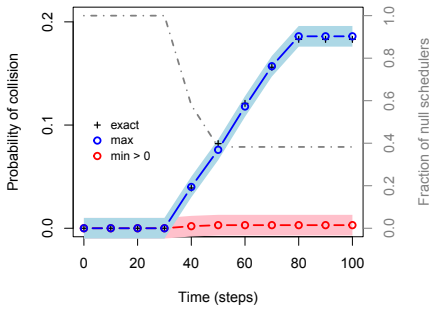

Figure 4. Max. and min. probabilities of second collision in WLAN protocol.

To demonstrate the scalability of our approach, we consider the choice coordination model of [17] and estimate the minimum probability that a group of six tourists will meet within T steps. The model has a parameter ($BOUND$) that limits the state space. We set $BOUND = 100$, making the state space of $\approx 5 \times 10^{16}$ intractable to numerical model checking. For $T = 20$ and $T = 25$ the true minimum probabilities are respectively 0.5 and 0.75. Using smart sampling and a Chernoff bound of $\varepsilon = \delta = 0.01$, we correctly estimate the probabilities to be 0.496 and 0.745 in a few tens of minutes on a standard laptop computer.

5 Prospects and Challenges

Our techniques are immediately extensible to continuous time MDPs and other models that use nondeterminism. It is also seems simple to consider MDPs with rewards. Although the presented algorithms are not optimised with respect to simulation budget, in a forthcoming work we introduce the notion of "smart sampling" to maximise the chance of finding good schedulers with a finite budget.

A limitation of our approach is that the algorithms sample from only a subset of possible schedulers. It is easy to construct examples where good schedulers are vanishingly rare and will not be found. Our ongoing focus is therefore to develop memory-efficient learning techniques that construct schedulers piecewise, to improve convergence and consider a much larger set of schedulers.

Acknowledgement This work was partially supported by the European Union Seventh Framework Programme under grant agreement no. 295261 (MEALS).

References

1. PLASMA project web page. https://project.inria.fr/plasma-lab/.
2. C. Baier and J.-P. Katoen. *Principles of model checking*. MIT Press, 2008.
3. R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.
4. A. Bianco and L. De Alfaro. Model checking of probabilistic and nondeterministic systems. In *Foundations of Software Technology and Theoretical Computer Science*, pages 499–513. Springer, 1995.
5. J. Bogdoll, L. M. F. Fioriti, A. Hartmanns, and H. Hermanns. Partial order methods for statistical model checking and simulation. In *Formal Techniques for Distributed Systems*, pages 59–74. Springer, 2011.
6. H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Ann. Math. Statist.*, 23(4):493–507, 1952.
7. E. Clarke, E. A. Emerson, and J. Sifakis. Model checking: algorithmic verification and debugging. *Commun. ACM*, 52(11):74–84, Nov. 2009.
8. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 3rd edition, 2009.
9. H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal aspects of computing*, 6(5):512–535, 1994.
10. A. Hartmanns and M. Timmer. On-the-fly confluence detection for statistical model checking. In *NASA Formal Methods*, pages 337–351. Springer, 2013.
11. D. Henriques, J. G. Martins, P. Zuliani, A. Platzer, and E. M. Clarke. Statistical model checking for Markov decision processes. In *Quantitative Evaluation of Systems, 2012 Ninth International Conference on*, pages 84–93. IEEE, 2012.
12. W. G. Horner. A new method of solving numerical equations of all orders, by continuous approximation. *Philosophical Transactions of the Royal Society of London*, 109:308–335, 1819.
13. M. Kearns, Y. Mansour, and A. Y. Ng. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine Learning*, 49(2-3):193–208, 2002.
14. D. E. Knuth. *The Art of Computer Programming*. Addison-Wesley, 3rd edition, 1998.
15. M. Z. Kwiatkowska, G. Norman, and J. Sproston. Probabilistic Model Checking of the IEEE 802.11 Wireless Local Area Network Protocol. In *Proc.* 2^{nd} *Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification*, pages 169–187. Springer-Verlag, 2002.
16. R. Lassaigne and S. Peyronnet. Approximate planning and verification for large Markov decision processes. In *Proc.* 27^{th} *Annual ACM Symposium on Applied Computing*, pages 1314–1319. ACM, 2012.
17. U. Ndukwu and A. McIver. An expectation transformer approach to predicate abstraction and data independence for probabilistic programs. In *Proc. 8th Workshop on Quantitative Aspects of Programming Languages (QAPL'10)*, 2010.
18. M. Okamoto. Some inequalities relating to the partial sum of binomial probabilities. *Annals of the Institute of Statistical Mathematics*, 10(1):29–35, 1958.
19. M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, 1994.
20. A. Wald. Sequential tests of statistical hypotheses. *The Annals of Mathematical Statistics*, 16(2):117–186, 1945.
21. H. L. S. Younes and R. G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *Computer Aided Verification*, pages 223–235. Springer, 2002.